ディープラーニングに基づくソフトウェア故障発生時間間隔の推定法 (確率的環境下における数理モデルの理論と応用)

AUTHOR(S):

田村, 慶信; 芦田, 哲; 松本, 光穂; 山田, 茂

CITATION:

田村, 慶信 ...[et al]. ディープラーニングに基づくソフトウェア故障発生時間間隔の推定法 (確率的環境下における数理モデルの理論と応用). 数理解析研究所講究録 2017, 2044: 52-60

ISSUE DATE:

2017-09

URL:

http://hdl.handle.net/2433/236981

数理解析研究所講究録
第2044巻 2017年 52-60

ディープラーニングに基づく ソフトウェア故障発生時間間隔の推定法

山口大学大学院・創成科学研究科 田村 慶信（Yoshinobu Tamura）†
†Graduate School of Sciences and Technology for Innovation, Yamaguchi University
山口大学大学院・理工学研究科 芦田哲（Satoshi Ashida）††
††Graduate School of Science and Engineering, Yamaguchi University
鳥取大学大学院・工学研究科 松本光穂（Mitsuho Matsumoto）†††
鳥取大学大学院・工学研究科 山田 茂（Shigeru Yamada）†††
†††Graduate School of Engineering, Tottori University

1 はじめに

短納期，コスト削減，標準化といった観点から，ソフトウェア開発現場においてもオープンソースソフトウェアが積極的に利用されている．特に，オープンソースソフトウェアのソースコードを理解できるだけでなく，その利活用まで可能なソフトウェア開発技術者の人材不足が指摘されている一方で，今後もオープンソースソフトウェアを利用したソフトウェア開発形態が多くなってくるものと思われる．近年では，企業主導の下でオープンソースプロジェクトが管理されるケースも増加しており，そのようなオープンソースソフトウェアでは一定の品質が維持されているケースも存在している．しかしながら，オープンソースソフトウェアの開発形態には，明確なテスト工程が存在しておらず，品質上の問題が指摘されている．多くのオープンソースソフトウェアでは，依然バグトラッキングシステムに基づいて品質上のデータ管理のみが行われているのが現状である．

ソフトウェアの信頼性を評価するための数理モデルとして，これまでに数百におよぶソフトウェア信頼性モデルが提案されてきた [1–3]．しかしながら，既存のソフトウェア信頼性モデルの多くは，オープンソースソフトウェア特有の開発環境の変化に伴うフォールト発見事象が考慮されていない．また，オープンソースソフトウェアに対して既存のソフトウェア信頼性モデルを適用することは可能であっても，オープンソースソフトウェア特有の開発環境やプロジェクト形態などに関する新たな知見を得ることはできない．これまでに，オープンソースソフトウェアを対象としたソフトウェア信頼性モデルに関する研究も行われているが [4]，既存のソフトウェア信頼性モデルを適用するためには，バグトラッキングシステム上に登録されているデータから，ソフトウェア信頼性モデルに適用可能な累積フォールト発見数データなどに編集する必要があった．さらに，バグトラッキングシステム上には，修正されたフォールトや未修正のフォールトなど様々なデータが混在しており，こうしたデータの全てをソフトウェア信頼性モデルに適用可能なように抽出する作業も必要となる．オープンソースソフトウェアの開発現場でソフトウェア信頼性モデルを適用することは，実利用上の観点から考えた場合，ソフトウェア信頼性モデルに関する知識のある開発管理者でなければ難しいといえる．バグトラッキングシステム上に登録されているデータをそのまま利用した品質評価と管理が可能となれば，開発管理者だけではなく利用者側にとっても有益となる．特に，バグトラッキングシステム上には，開発管理者によって登録されたデータだけではなく，一般ユーザによって報告されたデータも登録されている．全てのデータが，オープンソースソフトウェアに精通したプロジェクト管理者によって登録・管理されていれば問題はないが，例えば，フォールト

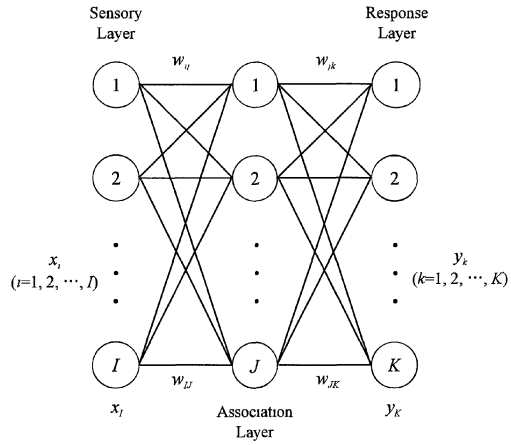

図 1： ニューラルネットワークの構成.

重要度やコンポーネントといったような分類データは，一般ユーザによって登録されたあとに修正されるケースもあり，様々な登録者が存在しているため，誤ったデータが存在しているケースも少なくない．

本論文では，ディープラーニング（deep learning）に基づくオープンソースソフトウェアに対する信頼性評価法を提案する．特に，実際のオープンソースソフトウェアのバグトラッキングシステム上に登録されているデータから，ソフトウェア信頼度成長傾向の予測手法について考察する．また，バグトラッキングシステム上に登録されているデータを利用した提案手法の具体的な数値例を示す．さらに，ディープラーニングに基づく提案手法と，既存の類似手法であるニューラルネットワークに基づく手法との，フォールト識別法に関する認識率について比較する．

2 ニューラルネットワークに基づくフォールト認識

本論文では，簡単のために図 1 に示すような，3 層ニューラルネットワークを適用する．まず，$w_{ij}^1(i = 1,2,\cdots,I; j = 1,2,\cdots,J)$ を入力層と中間層の結合係数，また $w_{jk}^2(j = 1,2,\cdots,J; k = 1,2,\cdots,K)$ は中間層と出力層の結合係数とする．さらに，$x_i(i = 1,2,\cdots,I)$ は入力データを表し，ニューラルネットワークの特徴量を考慮するために，以下に示すようなデータ $x_i(i = 1,2,\cdots,I)$ を適用する．

- バグトラッキングシステム上へ登録された日付
- ソフトウェアプロダクト名
- ソフトウェアコンポーネント名
- ソフトウェアバージョン名およびバージョン番号
- フォールト報告者のニックネーム
- フォールト修正者のニックネーム
- フォールトの状態
- フォールト重要度

入力層，中間層，出力層におけるユニットの数を，それぞれ I 個，J 個，および K 個とする．また，各層のユニットを示すインデックスを i, j, および k とする．ここで，各層のユニットの出力を h_j, y_k と

すると，

$$h_j = f\left(\sum_{i=1}^{I} w_{ij}^1 x_i\right), \tag{1}$$

$$y_k = f\left(\sum_{j=1}^{J} w_{jk}^2 h_j\right), \tag{2}$$

となる．但し，$f(\cdot)$ はシグモイド型関数であり，

$$f(x) = \frac{1}{1+e^{-\theta x}}, \tag{3}$$

として表される．ここで，θ はしきい値と呼ばれる定数である．ネットワークの学習を行うために，誤差逆伝播法を用いる [5]．ニューラルネットワークの出力層における値を $y_k(k=1,2,\cdots,K)$ とし，教師パターンを $d_k(k=1,2,\cdots,K)$ とすると，式 (2) の y_k は次式で評価される．

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - d_k)^2. \tag{4}$$

ここで，教師パターン $d_k(k=1,2,\cdots,K)$ には，信頼度成長傾向を表す成長係数 $d_k(k=1,2)$ を採用する．目的変数としての信頼度成長係数を表す変動値 $(1,-1)$ を定義する．例えば，変動値が 1 である場合は信頼度成長を，変動値が -1 である場合は信頼度退化を意味する．すなわち，数値化された入力データに基づいて，信頼度成長の判別に至るまでの結合状態の特徴をニューラルネットワークの結合係数に蓄積させ，信頼度成長の判別・予測が可能なモデルを考える．式 (4) の評価基準をもとに，結合係数が最急降下法にて決定される．

3　ディープラーニングに基づくフォールト認識

本論文におけるディープラーニングの構造を図 2 に示す．図 2 において，$z_l(l=1,2,\cdots,L)$ および $z_m(m=1,2,\cdots,M)$ は特徴抽出のための結合ユニットを意味する．また，$o_n(n=1,2,\cdots,N)$ は，圧縮された特徴量を表す．ディープラーニングに関しては，いくつかのアルゴリズムが提案されているが，本論文では，オープンソースプロジェクトにおけるバグトラッキングシステム上におけるフォールトデータを分析するために，ディープニューラルネットワークを採用する [6–11]．

ニューラルネットワークの場合と同様に，特徴抽出のための結合ユニットにおけるパラメータに対する情報量として以下のようなデータを適用する．このとき，9 種類の説明変数に対して，2 つの目的変数が最終的な特徴量として抽出されるように学習され，各入力データは出現頻度のような数値データに変換される．

- バグトラッキングシステム上へ登録された日付
- ソフトウェアプロダクト名
- ソフトウェアコンポーネント名
- ソフトウェアバージョン名およびバージョン番号
- フォールト報告者のニックネーム
- フォールト修正者のニックネーム
- フォールトの状態
- オペレーティングシステム名
- フォールト重要度

ディープラーニングの場合も同様に，目的変数としての信頼度成長係数を表す変動値 $(1,-1)$ を定義し，変動値が 1 である場合は信頼度成長を，変動値が-1 である場合は信頼度退化するものとし，過去のデータから信頼度成長係数を表す変動値を予測する．

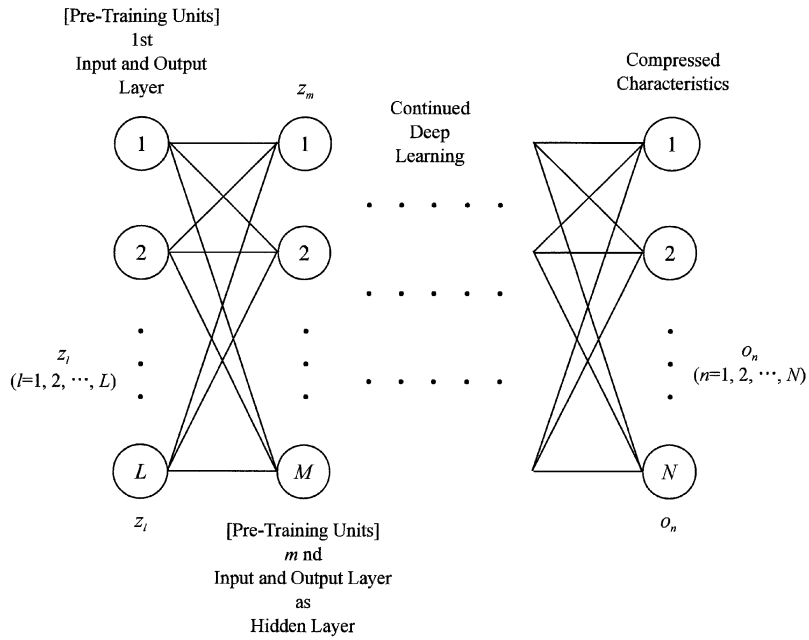

図 2： ディープラーニングの構造.

4 ソフトウェア信頼性評価尺度

平均ソフトウェア故障時間間隔（Mean Time between Software Failures，以下 MTBF を略す）は，ソフトウェアの故障発生頻度を表す有用な尺度として知られている．本論文では，k 番目のソフトウェア故障に対する MTBF を，

$$t_k = t_{k-1} + dl_{k-1} \cdot SD_{k-1}, \tag{5}$$

により定義する．ここで，dl_{k-1} はディープラーニングにより推定された $k-1$ 番目の信頼度成長係数としての変動値を表す．例えば，変動値が 1 である場合は信頼度成長を，変動値が-1 である場合は信頼度退化を意味する．また，SD_{k-1} は $k-1$ 番目の標準偏差であり，

$$SD_k = \sqrt{\frac{1}{k-1}\sum_{n=1}^{k}\left(t_n - \frac{1}{n}\sum_{l=1}^{n} t_l\right)^2}, \tag{6}$$

として与えられる．

さらに，信頼度成長傾向を把握するため，以下のような移動平均に基づく MTBF を用いる．

$$t_l^m = t_{l-1}^m + dl_{l-1}^m \cdot SD_{l-1}. \tag{7}$$

ここで，t_{l-1}^m は $l-1$ 番目の実測値に対するフォールト数 m 個の単純移動平均値を表す．また，dl_{l-1}^m はディープラーニングにより推定された $l-1$ 番目における変動値を表す．

5 数値例

数多くのオープンソースソフトウェアが世界中で開発・公開されている．オープンソースソフトウェアを利用することにより，コスト削減や短納期につながることから，様々な組織において積極的に利用さ

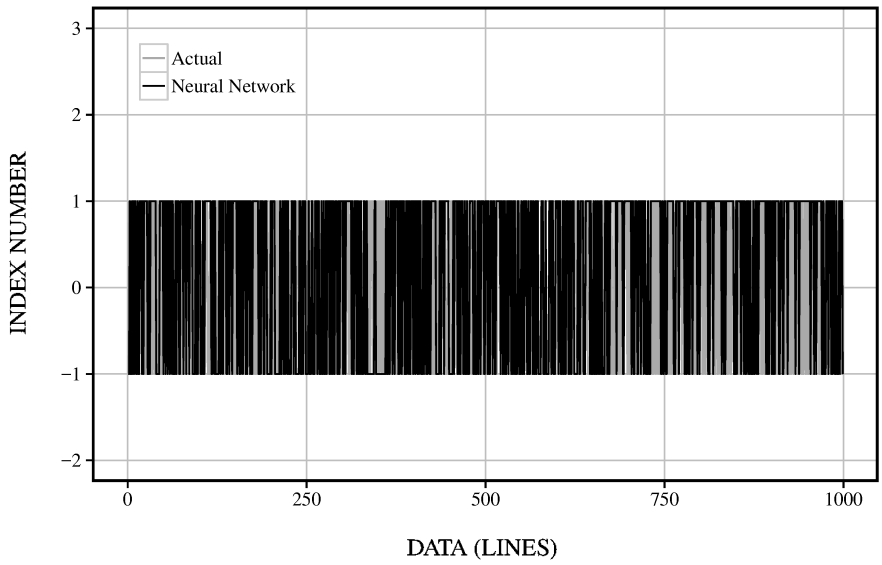

図 3： 90%の学習データに対するニューラルネットワークに基づく推定結果.

れている．また，現在では，組込みソフトウェア，オペレーティングシステム，アプリケーションソフトウェア，サーバソフトウェア，さらにはクラウドまで，多くの種類のオープンソースソフトウェアが開発されており，その多くが実社会において利用されている．本論文では，HTTP サーバソフトウェアとして開発されており，世界中で多用されている Apache HTTP サーバ [12] を例に取り上げる．Apache HTTP サーバの開発を行っている Apache Foundation のバグトラッキングシステム上に登録された 10,000 行のフォールトデータを収集した．このとき，提案手法の数値例を以下に示す．

5.1 信頼度成長傾向の認識率に関する推定結果

信頼度成長傾向に対する認識率の推定結果について考察する．まず，9,000 行の学習データを利用したニューラルネットワークに基づく 1,000 行のテストデータに対する推定結果を図 3 に示す．同様に，9,000 行の学習データを利用したディープラーニングに基づく 1,000 行のテストデータに対する推定結果を図 4 に示す．図 3 および図 4 から，ディープラーニングによる推定結果が，ニューラルネットワークに基づく推定結果に比べて良好な様子が確認できる．

さらに，90%の学習データを利用した場合におけるニューラルネットワークとディープラーニングに基づく推定結果に対する認識率の比較結果を表 1 に示す．表 1 から，ニューラルネットワークよりもディープラーニングに基づくフォールト識別法の方が認識率がやや高いことが確認できる．

5.2 信頼性評価尺度

推定された MTBF を図 5 に示す．図 5 から，$k-1$ 番目のソフトウェア故障に対する MTBF から，k 番目のソフトウェア故障に対する MTBF が推定されている様子が確認できる．また，10 個移動平均に基づく MTBF の推定結果を図 6 に示す．図 6 から，フォールトが発見されるにつれて MTBF の値が増加していく，すなわち信頼度成長が起こっている様子が確認できる．さらに，長期的な傾向を把握するために 50 個移動平均に基づく MTBF を図 7 に示す．図 7 から全体的に信頼度成長傾向であることが分かる．

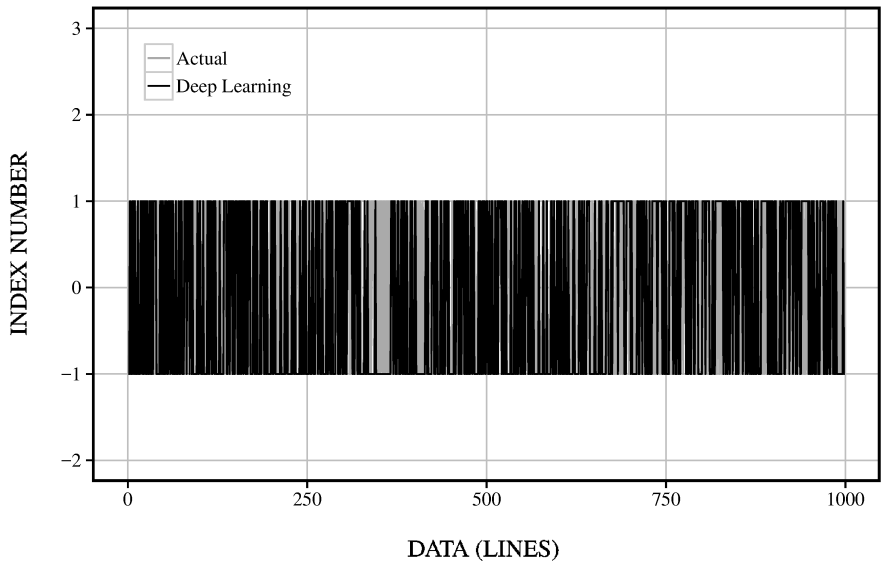

図 4： 90%の学習データに対するディープラーニングに基づく推定結果.

表 1： 90%の学習データを利用した場合におけるニューラルネットワークとディープラーニングに基づく推定結果に対する認識率の比較結果.

Estimation Method	Recognition Rate
Neural Network	71.57157%
Deep Learning	72.67267%

6 おわりに

オープンソースプロジェクトのバグトラッキングシステム上には，多くのフォールトデータが蓄積されている．これらのフォールトデータを効率良く利用することができるようになれば，オープンソースソフトウェアの品質向上に大きく寄与する．さらに，バグトラッキングシステム上へ登録されたフォールトに対して，事前に発生原因を特定することにより，迅速な修正対応が可能となるものと考えられる．

本論文では，ディープラーニングに基づくオープンソースソフトウェアに対する信頼性評価法を提案した．特に，実際のオープンソースソフトウェアのバグトラッキングシステム上に登録されているデータから，ソフトウェア信頼度の成長傾向を予測するための手法について考察した．また，バグトラッキングシステム上に登録されているデータを利用した提案手法に基づく数値例を示した．さらに，ディープラーニングに基づく提案手法と，既存の類似手法であるニューラルネットワークに基づく手法との，フォールト識別法に関する認識率について比較した．これらの結果から，提案手法の有効性が確認できた．特に，ディープラーニングに基づく手法は，その特性値を事前に抽出する必要がないため，バグトラッキングシステム上へ組み込むことにより，現存するデータをそのまま利用して認識することが可能となる．今後は，認識率向上に向けて，多くのデータセットにより検証を行う必要があるものと考える．

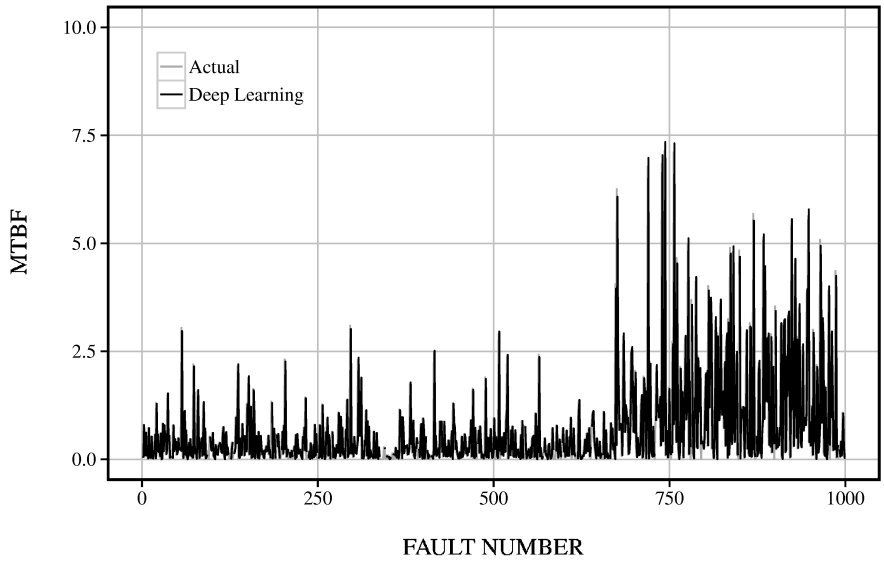

図 5： 推定された MTBF.

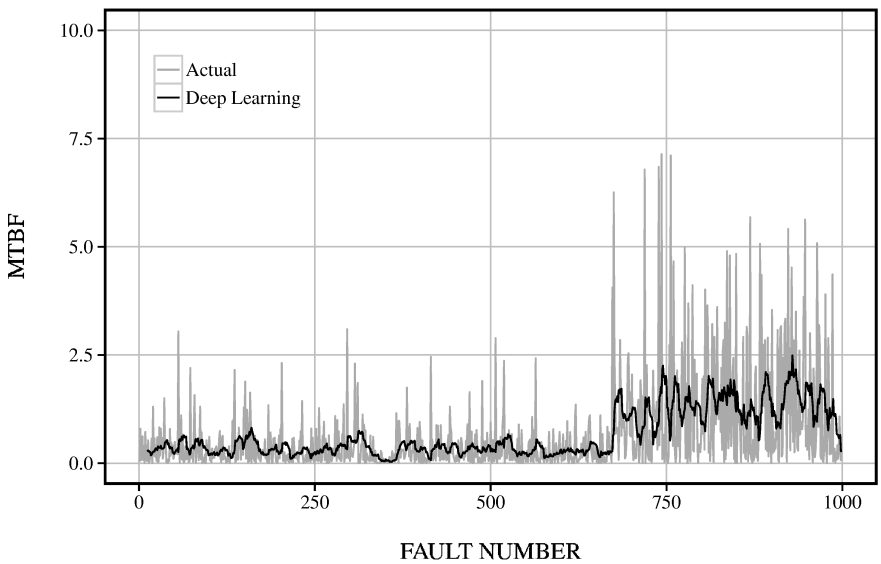

図 6： 推定された 10 個移動平均に基づく MTBF.

謝辞

本研究の一部は，公益財団法人 電気通信普及財団調査研究助成，公益財団法人 大川情報通信基金，および JSPS 科研費基盤研究（C）（課題番号 15K00102 および 16K01242）の援助を受けたことを付記する．

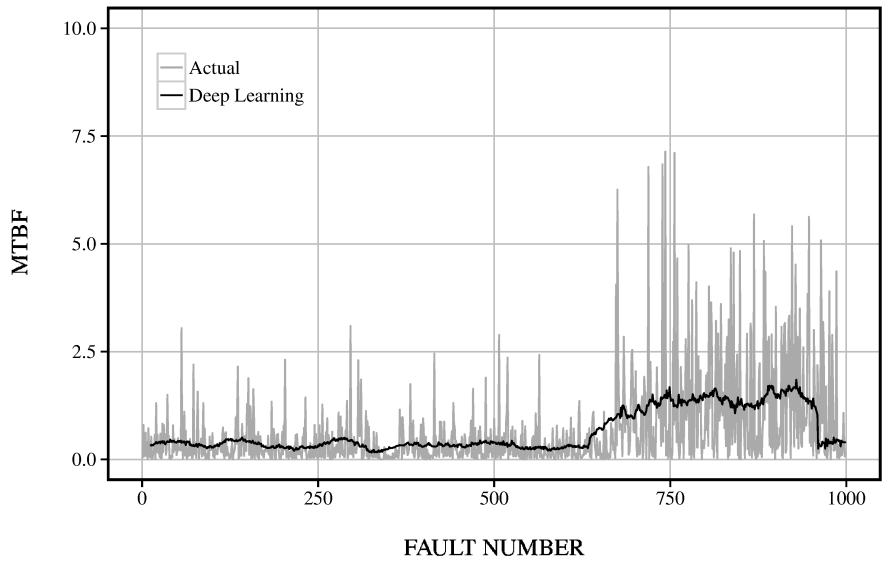

図 7： 推定された 50 個移動平均に基づく MTBF.

参考文献

[1] M.R. Lyu, ed., *Handbook of Software Reliability Engineering*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[2] S. Yamada, *Software Reliability Modeling: Fundamentals and Applications*, Springer–Verlag, Tokyo/Heidelberg, 2014.

[3] P.K. Kapur, H. Pham, A. Gupta, and P.C. Jha, *Software Reliability Assessment with OR Applications*, Springer–Verlag, London, 2011.

[4] S. Yamada and Y. Tamura, *OSS Reliability Measurement and Assessment*, Springer–Verlag, London, 2016.

[5] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transactions on Neural Networks.*, Vol. 1, pp. 239-242, June 1990.

[6] D.P. Kingma, D.J. Rezende, S. Mohamed, and M. Welling, "Semi-supervised learning with deep generative models," *Proceedings of Neural Information Processing Systems*, 2014, pp. 3581–3589.

[7] A. Blum, J. Lafferty, M.R. Rwebangira, and R. Reddy, "Semi-supervised learning using randomized mincuts," *Proceedings of the International Conference on Machine Learning*, 2004, pp. 1–13.

[8] E.D. George, Y. Dong, D. Li, and A. Alex, "Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition," *IEEE Transactions on Audio, Speech, and Language Processing*, Vol. 20, No. 1, pp.30-42, 2012.

[9] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, Vol. 11, No. 2, pp. 3371–3408, 2010.

[10] H.P. Martinez, Y. Bengio, and G.N. Yannakakis, "Learning deep physiological models of affect," *IEEE Computational Intelligence Magazine*, Vol. 8, No. 2, pp. 20–33, 2013.

[11] B. Hutchinson, L. Deng, and D. Yu, "Tensor deep stacking networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 8, pp. 1944–1957, 2013.

[12] The Apache Software Foundation,
The Apache HTTP Server Project, http://httpd.apache.org/